Swarm Intelligence Course (INFO-H-414)
Exams

May 12, 2017

1 Modalities

The exam is divided in two parts:

Project You can choose a project among the ones proposed below. You are asked to provide the required deliverables and to present your project in a 7-minute talk, followed by 5 minutes of questions. This will account for up to 10 points of your final grade.

Questions You will be asked a number of questions concerning *the entire course material*. This will account for up to 10 points of your final grade.

To pass the exam one must obtain at least 5 points in each part of the exam.

1.1 Timeline

- The date of the exam is June 13th.
- The project submission deadline is June 6th at 23:59. If this deadline is missed, you go to the second session.
- Email your choice (ACO or Swarm Robotics) to both the responsibles of the topic of the project you choose (see contacts at the end of the document).

1.2 Project Deliverables

For the project, you will have to provide:

- Your code in digital format (i.e., text files), so we can test it. Send it by e-mail to the people responsible for your project (see contacts at the end of the document);
- A short document (6-8 pages, single spaced, font size 12) written in English that describes your work. You have to explain both the idea and the implementation of your solution, follow the instructions provided for each project.

- On the day of the oral presentation, you are expected to show slides and come with your own laptop. If you happen to not have a laptop, send us a PDF version of your slides **before** the day of the exam.

2 Projects

2.1 General Remarks

Apply what you learned It is mostly important that you stick to the principles of swarm intelligence: simple, local interactions, no global communication, no global information.

Avoid complexity If your solution gets too complex, it is because it is the wrong one.

Honesty pays off If you find the solution in a paper or in a website, cite it and say why you used that idea and how.

Cooperation is forbidden Always remember that this is an **individual** work.

The project counts for 50% of your final grade. The basic precondition for you to succeed in the project is that it must work. If it does not, the project will not be considered sufficient. In addition, code must be understandable — comments are mandatory.

The document is very important too. We will evaluate the quality of your text, its clarity, its completeness, and its soundness. References to existing methods are considered a plus — honesty *does* pay off! More specifically, the document is good if it contains all the information needed to reproduce your work without having seen the code and a good and complete analysis of the results.

The oral presentation is also very important. In contrast to the document, a good talk deals with *ideas* and leaves the technical details out. Be sure that it fits in the 7-minute slot.

2.2 Ant Colony Optimization

For this project your are required to design, implement, evaluate and analyse the use of ACO algorithms for the *Closest String Problem* (CSP). The Closest String Problem is an NP-hard combinatorial optimization problem. It has applications in different fields such as molecular biology, where it can be used in the design of genetic drugs with structure similar to a set of existing sequences of RNA, and coding theory, where it can be applied in message encoding when searching sequences of characters which are closest to some given set of strings.

The CSP can be stated as follows: we define an alphabet $\mathcal{A} = \{c^1, \ldots, c^m\}$ composed of m characters and a set $\mathcal{S} = \{s^1, \ldots, s^n\}$ of n strings, each of them containing ℓ characters of $\mathcal{A}$. The distance between two strings is measured with

the *Hamming distance*. Given the function $h(s_i, t_i)$ that takes two characters s_i and t_i:

$$h(s_i, t_i) = \begin{cases} 0, & s_i = t_i \\ 1, & s_i \neq t_i \end{cases} \tag{1}$$

The Hamming distance between two strings s and t is defined as follows:

$$\mathrm{HD}(s, t) = \sum_{i=0}^{\ell} h(s_i, t_i) \tag{2}$$

For example, given the set $\mathcal{S} = \{\mathrm{AGTC}, \mathrm{AATG}, \mathrm{CATG}\}$ and the string $t = \mathrm{AATC}$ has distances 1, 1 and 2 to the strings in $\mathcal{S}$ respectively. The objective of the CSP is to find a string t of size ℓ that minimizes the maximum distance (d) to any string in the set $\mathcal{S}$:

$$\arg\min f(t) = \{t \mid f(t) = d = \max_{s \in \mathcal{S}}(HD(s, t))\} \tag{3}$$

In the previous example, the evaluation of the string is $f(t) = d = 2$. The are no constraints on how to combine the characters of $\mathcal{A}$ to form a string.

2.2.1 Goal

You have to implement two ACO algorithm variants, test your implementations and write a report about your work analyzing the results. You can re-use the ACO implementations developed in class, note that adaptations are needed to solve the CSP problem. You are free to define the components of the algorithms as you think best, as long as they follow the general definition of each algorithm. The implementation of ideas proposed in the literature (citation should be included) will be rewarded. Once the algorithms are implemented, experiments should be carried out in order to compare and analyze their performance. Finally, the addition of local search to the ACO algorithms should be studied. You are free to propose any kind of local search for the problem. Starting points for CSP in the literature are [3], [2] and [6]. You are free to use ideas of the literature as long as the source is properly cited in your report.

2.2.2 Instances

Benchmark instances for the CSP can be found at `http://iridia.ulb.ac.be/~lperez/INFO-H-414/project/instances.tar.gz`. We provide instances of different size and type, taken the set of random instances available in [1]. The format of each instance line is the following:

1. alphabet size (k)
2. number of strings (n)
3. string length (ℓ)
4. k lines containing each one character of the alphabet

5. n lines containing the strings

An example of instance file is the following:

data.txt

```
4
3
7
A
C
G
T
AAGTGTA
DAGTGGA
ACGTGTT
```

2.2.3 Deliverables

The final deliverable report should include:

- Report (pdf format)
- Source code
- Results in *csv* format

Please follow this guideline for you deliverable:

Report:

1. Implementation of two ACO algorithm variants for solving the CSP
   (a) Describe the implemented algorithms. Please explain only the relevant design decisions (representation of solutions, pheromone and heuristic information definition, update mechanism, transition rule, etc.).
   (b) Set the parameters of the ACO algorithms and report the best parameter settings for each algorithm. Describe the process you used to obtain the parameter settings. If you use automatic tuning, please report: parameters tuned, set of values (domain) for each parameter and number of evaluations allowed for the tuner (some freely available tuners are [5] and [4]).
   (c) For each algorithm, perform 10 runs on each instance of the instance set.
   Note1.: you **must** use the same budget (evaluations) for both algorithms.
   Note2.: you **must** use paired seeds for the 10 runs as explained in class.

(d) For each instance/algorithm report: best (B), worst (W), mean (M) and standard deviation (SD) of the 10 runs in a *csv* file (algorithm_name-results.cvs), where each row is an instance.

(e) Compare the performance of the algorithms in the report:

- Compare the results calculating the relative percentage deviation to the upper bound provided in the `instances_opt.txt` file:

$$rpd = \frac{(x - upb) * 100}{upb} \tag{4}$$

- Use plots and/or tables to show the results. Aggregate the results of all instances.
- Perform a pairwise comparison of the algorithms using the Wilcoxon rank-sum test.
- Based on the results, which of the implemented algorithms would you recommend for solving this problem? Comment.
- Compare the convergence of the algorithms.
  (Note: The goal is to compare the computation effort (evaluations) vs. solution quality).

2. Choose an algorithm and add local search search to it:
   (a) Describe the local search procedure implemented.
   (b) Perform 10 runs per instance for the new algorithm and compare it with its version without local search. Report as above.
   (c) Are there any improvements? Why? Comment.
3. Tune the parameters of the algorithm with local search. Analyze your results, is the algorithm more sensitive to some parameters?

Code:

- The code should run in the same way as the code developed in class, that is, using the same command line arguments. Feel free to add new arguments in case you need them.
- The code should be properly commented and ordered (indent!).
- The code should include a README file with the main instructions and specifications of your algorithms and parameters.
- You must implement your own code. Plagiarism will be penalized.
- Make sure your implementation works on Linux.

2.3 Swarm Robotics: *Foraging with Collective Decision Making*

The activity of food search and retrieval is commonly referred to as *foraging*. In swarm robotics, foraging is a commonly used task to compare different algorithms for exploration (what is the best way to discover interesting places in

the environment?), division of labor (who should explore? for how long?), etc. In the most general setting, food items are scattered in an environment at locations unknown to the robots and the robots need to explore the environment, find the food, and take it to the nest.

In this project, we consider food items with different energy levels: one positive and one negative. The food is gathered in two areas of the environment according to their energy level. The location of the source that yields items with positive energy level is unknown to the robots. The students are asked to provide and study the performance of two robot swarms with the capability of discriminating between two foraging sites and collectively forage from the best one.

2.3.1 Problem definition

The robot swarm operates in a dodecagonal environment composed of two target sources and a nest. Each target source is represented as a circle with different quality value v_i determined as the ground color value, with $v_i \in [0.05, 0.95]$ and $i \in [0, 1]$. Each of the source possesses an unlimited number of food items. The darker source yields items with energy level of value 3 while the lighter one yields items with energy level of value -1. The robots can perceive the ground color through the ground sensor. In order for the robots to differentiate the two sources, a cylinder object marked with a red or blue LED is placed in the middle of each source. The robots can perceive the LEDs through the omnidirectional camera. A light is placed above the rectangular nest. The robots can perceive it with the light sensor and can use it to navigate in the environment.

Each experiment is automatically terminated after 1000 s (10000 time-steps).

Goal The goal of the robot swarm is to maximize the energy level E retrieved at the nest during the whole duration of the experiment, that is

$$\max E = N_0 e_0 + N_1 e_1 \quad \text{with} \begin{cases} e_0 = 3, e_1 = -1 & \text{if } v_0 < v_1 \\ e_0 = -1, e_1 = 3 & \text{if } v_0 > v_1 \end{cases} \tag{5}$$

with N_0 being the number of items collected from source 0 and N_1 the number of items collected from source 1.

Swarm composition The swarm comprises 30 homogeneous robots. The robots are equipped with the following sensors and actuators:

- **omnidirectional camera** to detect and distinguish between the sources (marked with LEDs);
- **light** to perceive the light above the nest;
- **ground sensor** to sense the color of the ground;
- **proximity** to avoid obstacles;

- **wheels** to explore the environment;
- **LEDs** to display information.

Additional Remarks

- ARGoS has been configured so that the robots collect items on sources and drop them at the nest automatically. A robot can only carry one item at a time. If a robot goes to source 1 then to source 0, the item collected at source 1 will be replaced by an item from source 0.
- The color used for the LED on top of the cylindrical objects that are placed in the middle of the sources is independent of the energy level of the source. The LEDs colors are fixed once and for all experiments.
- We assume that the robots have such a limited memory that they only can memorize the quality of the last visited source. Each robot that estimates the quality of a source must overwrite the previous quality estimate (if any) that it stored. Note that this constraint must not be violated in order to obtain a sufficient solution.

2.3.2 Goals

The goal of this project is to design, implement, test, and compare **two robot controllers** that try to maximize the energy value collected at the nest.

Many solutions are possible. The students should come up with two controllers that are conceptually different. Remember to follow the principles of swarm robotics and apply what you learned during the course. Solutions in which robots choose individually the best source to forage by comparing the quality of the two sources will be considered not sufficient.

ARGoS is configured to automatically dump data on a file whose name can be changed in the .argos experiment configuration. This file is composed of a line indicating which source has the high energy level food items, and a table containing four columns:

- The current step
- The number of items collected from source 0
- The number of items collected from source 1
- The total energy level E.

A complete analysis must be performed to evaluate and compare the quality of the two controllers implemented. In particular, to present statistically meaningful results, we suggest you to execute and collect the results over at least 30 runs of each controller. The 30 runs of a controller should differ from each other for the random seed specified in the .argos experiment. To ensure a fair

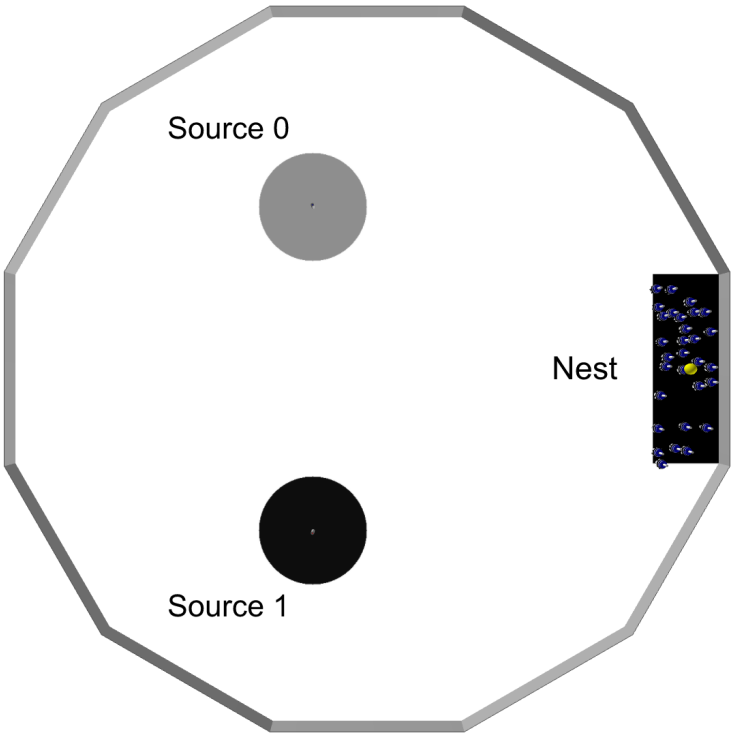

Figure 1: A top view of the environment. The nest is black and a light, visible from everywhere in the arena, is placed on top of it. Each source has a different quality which may vary in each experiment and is determined by the ground color. In this case, the best source is source 1 with $v_1 = 0.0519$.

comparison, you should use the same 30 random seeds for both controllers. In your report, you will include two tables, one for each controller implemented, which will contain the following columns:

- Random seed
- Final number of items collected from source 0
- Final number of items collected from source 1
- Final energy level E.

Beside these two tables, quantitative numerical measures of the performance of the two solutions must be produced. Specifically, you should produce i) plots that show the trend over time of the two controllers implemented—e.g. number of items collected from source 0 against number of items collected from source 1, ii) a plot that compares the performance of the two controllers, and iii) a

statistical significance test that proves whether the two controllers are significantly different. On the basis of these measures, you should discuss the results and draw the appropriate conclusions.

Be aware that, for the project evaluation, **the analysis is as important as the implementation**. A project presenting a controller that is capable of selecting the optimal source wonderfully will be evaluated poorly if the analysis part is missing or limited.

Setting up the code

- Download the experiment files: SR_Project_H414.tar from http://iridia.ulb.ac.be/~lgarattoni/h-414/SR_Project_H414.tar.
- Unpack the archive and compile the code:
  - $ tar xvf SR_Project_H414.tar # Unpacking
  - $ cd SR_Project_H414 # Enter the directory
  - $ mkdir build # Creating build dir
  - $ cd build # Entering build dir
  - $ cmake -DCMAKE_BUILD_TYPE=Release ../src # Configuring the build dir
  - $ make # Compiling the code
- Set the environment variable ARGOS_PLUGIN_PATH to the full path in which the build/ directory is located:

  $ export ARGOS_PLUGIN_PATH=/path/to/SR_Project_H414/build/
- You can also put this line into your $HOME/.bashrc file, so it will be automatically executed every time you open a console. Run the experiment to check that everything is OK:
  - $ cd /path/to/SR_Project_H414 # Make sure you are in the right directory
  - $ argos3 -c foraging.xml # Run the experiment

If the usual ARGoS GUI appears, you're ready to go.

2.4 Setting up the experiment

Switching the visualization on and off. The experiment configuration file allows you to launch ARGoS both with and without visualization. When you launch ARGoS with the visualization, you can program the robots interactively exactly like you did during the course. Launching ARGoS without the visualization allows you to run multiple repetitions of an experiment automatically, e.g., through a script. By default, the script launches ARGoS in interactive mode.

To switch the visualization off, just substitute the visualization section with: <visualization />, or, equivalently, comment out the entire qt-opengl section.

Loading a script at init time. When you launch ARGoS without visualization, you cannot use the GUI to set the running script. However, you can modify the XML configuration file to load automatically a script for you. At line 48 of foraging.argos you'll see that the Lua controller has an empty section <params />. An example of how to set the script is at line 51 of the same file. Just comment line 48, uncomment line 51 and set the script attribute to the file name of your script.

Changing the random seed. When you want to run multiple repetitions of an experiment, it is necessary to change the random seed every time. To change the random seed, set the value at line 11 of foraging.argos, attribute random_seed.

Changing the output file name. As explained above, ARGoS automatically dumps data to a file as the experiment goes. To set the name of this file, set a new value for the attribute output at line 17 of foraging.argos.

Making ARGoS run faster. Sometimes ARGoS is a little slow, especially when many robots and many sensors are being simulated. You can make ARGoS go faster by setting the attribute threads at line 9 of foraging.argos. Experiment with the values, because the best setting depends on your computer.

2.4.1 Deliverables

The final deliverables must include source code and documentation:

Code: The Lua scripts that you developed, well-commented and well-structured.

Documentation: A report of 6-8 pages structured as follows:

- Main idea of your approaches.
- Structure of your solutions (the state machines).
- Analysis and comparisons of the results.

3 Contacts

Marco Dorigo	mdorigo@ulb.ac.be	for general questions
Mauro Birattari	mbiro@ulb.ac.be	for general questions
Leslie Pérez Cáceres	leslie.perez.caceres@ulb.ac.be	for ACO
Hayfa Hammami	haifa.hammami@ulb.ac.be	for ACO
Lorenzo Garattoni	lgaratto@ulb.ac.be	for swarm robotics
Antoine Ligot	aligot@ulb.ac.be	for swarm robotics

References

[1] Closest string and closest substring problems. benchmark description. http://tcs.informatik.uos.de/research/csp_cssp. Accessed: 04-05-2017.

[2] Markus Chimani, Matthias Woste, and Sebastian Böcker. *A Closer Look at the Closest String and Closest Substring Problem*, pages 13–24. 2011.

[3] Simone Faro and Elisa Pappalardo. *Ant-CSP: An Ant Colony Optimization Algorithm for the Closest String Problem*, pages 370–381. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[4] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configurationframework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009.

[5] Manuel López-Ibá nez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[6] Elisa Pappalardo, Panos M. Pardalos, and Giovanni Stracquadanio. *Optimization Approaches for Solving String Selection Problems*, chapter 3. Springer-Verlag New York, 2013.